Matthew Ridder,
Joseph Fackler,
Doug Etts

Design Review 2: Team NOR

Design Review 2 Progress

In this second design sprint, we successfully completed the functionality for the ADD, SUB, and SHIFT components. We also completed the Register, and finished all of the top level connections for our ALU.

When researching designs for the Adder and Subtractor components, we noticed that they have very similar functionalities at the transistor level. Because of this, we decided to combine these components into one block using Cin to determine whether it adds or subtracts. When Cin is 0, the block adds two unsigned integers and Cout represents the carry bit. When Cin is 1, the block subtracts two integers using a 2's complement implementation and Cout acts as a sign flag. In this case, Cout=0 represents a positive number and Cout=1 represents a negative number. For this Add/Sub block, we decided to use a series of Mirror stages. We did this because the mirror stages gave us a smaller area cost and delay time than the Static CMOS implementation by avoiding large PMOS stacks and redundant inverters. To size our adder we just used minimum sizes. However, we developed a Matlab Script to automatically determine the transistor sizes using logical effort that we will utilize in the optimization phase of our project.

We decided to use a Barrel Shifter implementation for our Shifter because each bit would only have to pass through one transmission gate to get to the output. Although this implementation results in a larger area cost than other possible implementations, it excels at minimizing delay time. We decided that the tradeoff is worth it since delay is more important in the metric that is going to be used to judge our design (Metric = (Active Power)*Delay^2*Area). In our design for the Shifter, we also decided to include left and right shift functionality because we thought that this capability would be very useful to anyone using it since shifters are often used to multiply and divided by powers of 2. This fact makes the increase in functionality worth the increase in area cost.

We decided to use a static implementation for our Register instead of a dynamic implementation in order to avoid the risk of having our register lose its value after a certain amount of time. The static implementation we chose was a series 2-1 mux implementation because it was simple and didn't cost a large amount of area and delay.

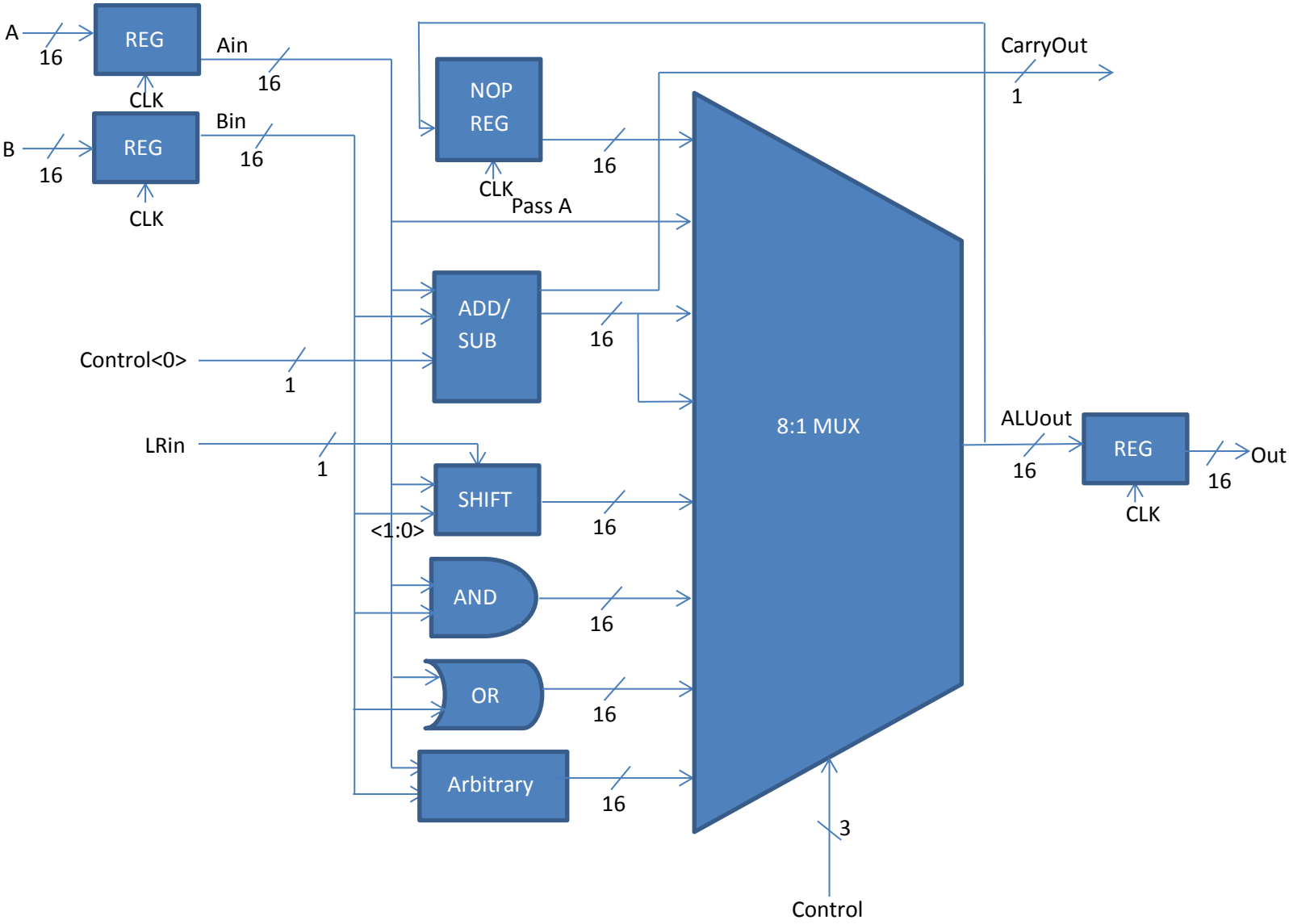
Remaining Tasks

At this point we have completed the expected functionality of each individual component in the ALU, except for the arbitrary function, and have made all of the top level connections for the ALU. However, we have not yet focused on optimizing the ALU for delay, area, and power. In the next few weeks before the finalized ALU is due, we will focus on those metrics using numerous sizing techniques and better design strategies. Also, we will need to finish building our arbitrary function. For that component, we have made a final decision and will create an 8-by-8 Wallace Tree Multiplier. After we

Matthew Ridder,
Joseph Fackler,
Doug Etts

have completed the functionality and optimization of our ALU, we will write the 4 page conference-style report and prepare to present it to the class.

Here is our updated Block Diagram for the ALU:



Matthew Ridder,
Joseph Fackler,
Doug Etts

Netlists for Frequently Used Components/Gate Level Netlists

```
// Cell name: TNOR_Inverter
// Function: 1-bit Inverter
// Inputs: in
// Outputs: out
subckt TNOR_Inverter VDD VSS in out
parameters wp=180n wn=90n ln=50n lp=50n mult=1
MP (out in VDD VDD) PMOS_VTL w=wp l=lp as=100n*wp ad=100n*wp
ps=200n+wp pd=200n+wp m=mult
MN (out in VSS VSS) NMOS_VTL w=wn l=ln as=100n*wn ad=100n*wn
ps=200n+wn pd=200n+wn m=mult
ends TNOR_Inverter
// End of subcircuit definition

// Cell Name: InvBuffer
// Function: Buffers input with a series of 2 inverters
// Input: A
// Output: out
subckt InvBuffer VDD VSS A out
I0 (VDD VSS A net0) TNOR_Inverter
I1 (VDD VSS net0 out) TNOR_Inverter
ends InvBuffer
// Ends subckt def

// Cell Name: TNOR_TX
// Function: Transmission Gate
// Input: in A
// Output: out
subckt TNOR_TX VDD VSS in A out
parameters wp=180n wn=90n lp=50n ln=50n mult=1
IO (VDD VSS A Abar) TNOR_Inverter
M0 (in A out out) NMOS_VTL w=wn l=ln as=100n*wn ad=100n*wn ps=200n+wn
pd=200n+wn m=mult
M1 (in Abar out out) PMOS_VTL w=wp l=lp as=100n*wp ad=100n*wp
ps=200n+wp pd=200n+wp m=mult
ends TNOR_TX

// Cell Name: TNOR_NAND2
// Function: 2-Input NAND
// Inputs: A B
// Output: out
subckt TNOR_NAND2 VDD VSS A B out
parameters wp=180n wn=90n lp=50n ln=50n mult=1
M0 (out A VDD VDD) PMOS_VTL w=wp l=lp as=100n*wp ad=100n*wp ps=200n+wp
pd=200n+wp m=mult
M1 (out B VDD VDD) PMOS_VTL w=wp l=lp as=100n*wp ad=100n*wp ps=200n+wp
pd=200n+wp m=mult
M2 (net5 B VSS VSS) NMOS_VTL w=wn l=ln as=100n*2*wn ad=100n*2*wn
ps=200n+2*wn pd=200n+2*wn m=mult
```

```
M3 (out A net5 VSS) NMOS_VTL w=wn l=ln as=100n*2*wn ad=100n*2*wn
ps=200n+2*wn pd=200n+2*wn m=mult
ends TNOR_NAND2
// end of subckt def

// Cell name: TNOR_AND2
// Function: 2-Input AND
// Inputs: A B
// Outputs: out
subckt TNOR_AND2 VDD VSS A B out
I0 (VDD VSS A B nandout) TNOR_NAND2
I1 (VDD VSS nandout out) TNOR_Inverter
ends TNOR_AND2
// End of subcircuit definition

// Cell name: TNOR_NOR2
// Function: 2-input NOR
// Inputs: A B
// Outputs: out
subckt TNOR_NOR2 VDD VSS A B out
parameters wp=180n wn=90n lp=50n ln=50n mult=1
M1 (out B VSS VSS) NMOS_VTL w=wn l=ln as=100n*wn ad=100n*wn ps=200n+wn
pd=200n+wn m=mult
M0 (out A VSS VSS) NMOS_VTL w=wn l=ln as=100n*wn ad=100n*wn ps=200n+wn
pd=200n+wn m=mult
M2 (out B net0 VDD) PMOS_VTL w=wp l=lp as=100n*2*wp ad=100n*2*wp
ps=200n+2*wp pd=200n+2*wp m=mult
M3 (net0 A VDD VDD) PMOS_VTL w=wp l=lp as=100n*2*wp ad=100n*2*wp
ps=200n+2*wp pd=200n+2*wp m=mult
ends TNOR_NOR2
// End of subcircuit definition.

// Cell name:TNOR_OR2
// Function: 2-input OR
// Inputs: A B
// Outputs: out
subckt TNOR_OR2 VDD VSS A B out
I0 (VDD VSS A B norout) TNOR_NOR2
I1 (VDD VSS norout out) TNOR_Inverter
ends TNOR_OR2
// End of subcircuit definition.

// Cell Name: TNOR_XOR
// Function: XOR gate that is used for the Add/Sub
// Inputs: A B
// Outputs: out
subckt TNOR_XOR VDD VSS A B out
parameters wp=180n wn=90n lp=50n ln=50n mult=1
MP0 (net0 A VDD VDD) PMOS_VTL w=wp l=lp as=100n*wp ad=100n*wp
ps=200n+wp pd=200n+wp m=mult
```

```
MP1 (out A B VDD) PMOS_VTL w=wp l=lp as=100n*wp ad=100n*wp ps=200n+wp
pd=200n+wp m=mult
MP2 (out B A VDD) PMOS_VTL w=wp l=lp as=100n*wp ad=100n*wp ps=200n+wp
pd=200n+wp m=mult
MN0 (net0 A VSS VSS) NMOS_VTL w=wn l=ln as=100n*wn ad=100n*wn
ps=200n+wn pd=200n+wn m=mult
MN1 (out net0 B VSS) NMOS_VTL w=wn l=ln as=100n*wn ad=100n*wn
ps=200n+wn pd=200n+wn m=mult
MN2 (out B net0 VSS) NMOS_VTL w=wn l=ln as=100n*wn ad=100n*wn
ps=200n+wn pd=200n+wn m=mult
ends TNOR_XOR
// Ends subckt def

// Cell name: TNOR_NAND3
// Function: 3-Input NAND
// Inputs: A B C
// Output: OUT
subckt TNOR_NAND3 VDD VSS A B C OUT
parameters wp=180n wn=270n ln=50n lp=50n mult=1
M2 (net7 C VSS VSS) NMOS_VTL w=wn l=ln as=100n*2*wn ad=100n*2*wn
ps=200n+2*wn pd=200n+2*wn m=mult
M1 (OUT A net15 VSS) NMOS_VTL w=wn l=ln as=100n*2*wn ad=100n*2*wn
ps=200n+2*wn pd=200n+2*wn m=mult
M0 (net15 B net7 VSS) NMOS_VTL w=wn l=ln as=100n*2*wn ad=100n*2*wn
ps=200n+2*wn pd=200n+2*wn m=mult
M5 (OUT C VDD VDD) PMOS_VTL w=wp l=lp as=100n*wp ad=100n*wp ps=200n+wp
pd=200n+wp m=mult
M4 (OUT B VDD VDD) PMOS_VTL w=wp l=lp as=100n*wp ad=100n*wp ps=200n+wp
pd=200n+wp m=mult
M3 (OUT A VDD VDD) PMOS_VTL w=wp l=lp as=100n*wp ad=100n*wp ps=200n+wp
pd=200n+wp m=mult
ends TNOR_NAND3
// End of subcircuit definition

// Cell name: TNOR_AND3
// Function: 3-Input AND
// Inputs: A B C
// Outputs: OUT
subckt TNOR_AND3 VDD VSS A B C OUT
  I1 (VDD VSS A B C nandout) TNOR_NAND3
  I0 (VDD VSS nandout OUT) TNOR_Inverter
ends TNOR_AND3
// End of subcircuit definition

// Cell name: TNOR_NAND4
// Function: 4-Input NAND
// Inputs: A B C D
// Ouputs: out
// View name: schematic
subckt TNOR_NAND4 VDD VSS A B C D out
```

```
parameters wp=90n wn=360n lp=50n ln=50n mult=1
M0 (net0 A VSS VSS) NMOS_VTL w=wn l=ln as=100n*4*wn ad=100n*4*wn
ps=200n+4*wn pd=200n+4*wn m=mult
M1 (net1 B net0 VSS) NMOS_VTL w=wn l=ln as=100n*4*wn ad=100n*4*wn
ps=200n+4*wn pd=200n+4*wn m=mult
M2 (net2 C net1 VSS) NMOS_VTL w=wn l=ln as=100n*4*wn ad=100n*4*wn
ps=200n+4*wn pd=200n+4*wn m=mult
M3 (out D net2 VSS) NMOS_VTL w=wn l=ln as=100n*4*wn ad=100n*4*wn
ps=200n+4*wn pd=200n+4*wn m=mult
M4 (out A VDD VDD) PMOS_VTL w=wp l=lp as=100n*wp ad=100n*wp ps=200n+wp
pd=200n+wp m=mult
M5 (out B VDD VDD) PMOS_VTL w=wp l=lp as=100n*wp ad=100n*wp ps=200n+wp
pd=200n+wp m=mult
M6 (out C VDD VDD) PMOS_VTL w=wp l=lp as=100n*wp ad=100n*wp ps=200n+wp
pd=200n+wp m=mult
M7 (out D VDD VDD) PMOS_VTL w=wp l=lp as=100n*wp ad=100n*wp ps=200n+wp
pd=200n+wp m=mult
ends TNOR_NAND4
// End of subcircuit definition for NAND

// Cell Name: Decoder
// Function: 3-8 Decoder, makes one line "hot"
// Input: in0 in1 in2
// Output: out0 out1 out2 out3 out4 out5 out6 out7
subckt Decoder VDD VSS in0 in1 in2 out0 out1 out2 out3 out4 out5 out6
out7
I0 (VDD VSS in0 inv0) TNOR_Inverter
I1 (VDD VSS in1 inv1) TNOR_Inverter
I2 (VDD VSS in2 inv2) TNOR_Inverter
I3 (VDD VSS inv0 inv1 inv2 out0) TNOR_AND3
I4 (VDD VSS in0 inv1 inv2 out1) TNOR_AND3
I5 (VDD VSS inv0 in1 inv2 out2) TNOR_AND3
I6 (VDD VSS in0 in1 inv2 out3) TNOR_AND3
I7 (VDD VSS inv0 inv1 in2 out4) TNOR_AND3
I8 (VDD VSS in0 inv1 in2 out5) TNOR_AND3
I9 (VDD VSS inv0 in1 in2 out6) TNOR_AND3
I10 (VDD VSS in0 in1 in2 out7) TNOR_AND3
ends Decoder

// Cell Name: TNOR_MUX2to1
// Function: 2:1 Multiplexer
// Inputs: A B S
// Output: out
// Relies on TNOR_Inverter and TNOR_NAND2
subckt TNOR_MUX2to1 VDD VSS A B S out
I0 (VDD VSS S net0) TNOR_Inverter
I1 (VDD VSS A net0 net1) TNOR_NAND2
I2 (VDD VSS B S net2) TNOR_NAND2
I3 (VDD VSS net1 net2 out) TNOR_NAND2
ends TNOR_MUX2to1
```

```
// End of subckt def

// Cell Name: TNOR_MUX4to1
// Function: 4:1 Multiplexer
// Inputs: A B C D S0 S1
// Output: out
// Relies on TNOR_Inverter, TNOR_NAND3, and TNOR_NAND4
subckt TNOR_MUX4to1 VDD VSS A B C D S0 S1 out
I0 (VDD VSS S0 net0) TNOR_Inverter
I1 (VDD VSS S1 net1) TNOR_Inverter
I2 (VDD VSS A net0 net1 net2) TNOR_NAND3
I3 (VDD VSS B S0 net1 net3) TNOR_NAND3
I4 (VDD VSS C net0 S1 net4) TNOR_NAND3
I5 (VDD VSS D S0 S1 net5) TNOR_NAND3
I6 (VDD VSS net2 net3 net4 net5 out) TNOR_NAND4
ends TNOR_MUX4to1
// Ends subckt def

// Cell Name: TNOR_MUX8to1
// 8:1 Multiplexer
// Inputs: A B C D E F G H S0 S1 S2
// Output: out
// Relies on TNOR_MUX2to1 and TNOR_MUX4to1
subckt TNOR_MUX8to1 VDD VSS A B C D E F G H S0 S1 S2 out
I0 (VDD VSS A B C D S0 S1 net0) TNOR_MUX4to1
I1 (VDD VSS E F G H S0 S1 net1) TNOR_MUX4to1
I2 (VDD VSS net0 net1 S2 out) TNOR_MUX2to1
ends TNOR_MUX8to1
// Ends subckt def

// Cell name: TNOR_REG1
// Function: 1-bit Static Register
// Inputs: D CLK
// Outputs: Q
subckt TNOR_REG1 VDD VSS D CLK Q
I0 (VDD VSS D bufin CLK bufin) TNOR_MUX2to1
I1 (VDD VSS CLK CLKB) TNOR_Inverter
I2 (VDD VSS bufin bufout) InvBuffer
I3 (VDD VSS bufout bufin2 CLKB bufin2) TNOR_MUX2to1
I4 (VDD VSS bufin2 Q) InvBuffer
ends TNOR_REG1
// End of subcircuit definition
```

Matthew Ridder,
Joseph Fackler,
Doug Etts

16-bit Register

```
// Cell name: TNOR_REG
// Function: 16-bit Register, static implementation
// Inputs: D<15:0> CLK
// Outputs: Q<15:0>
subckt TNOR_REG VDD VSS D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 D10 D11 \
       D12 D13 D14 D15 CLK Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 Q10 \
       Q11 Q12 Q13 Q14 Q15
I0 (VDD VSS D0 CLK Q0) TNOR_REG1
I1 (VDD VSS D1 CLK Q1) TNOR_REG1
I2 (VDD VSS D2 CLK Q2) TNOR_REG1
I3 (VDD VSS D3 CLK Q3) TNOR_REG1
I4 (VDD VSS D4 CLK Q4) TNOR_REG1
I5 (VDD VSS D5 CLK Q5) TNOR_REG1
I6 (VDD VSS D6 CLK Q6) TNOR_REG1
I7 (VDD VSS D7 CLK Q7) TNOR_REG1
I8 (VDD VSS D8 CLK Q8) TNOR_REG1
I9 (VDD VSS D9 CLK Q9) TNOR_REG1
I10 (VDD VSS D10 CLK Q10) TNOR_REG1
I11 (VDD VSS D11 CLK Q11) TNOR_REG1
I12 (VDD VSS D12 CLK Q12) TNOR_REG1
I13 (VDD VSS D13 CLK Q13) TNOR_REG1
I14 (VDD VSS D14 CLK Q14) TNOR_REG1
I15 (VDD VSS D15 CLK Q15) TNOR_REG1
ends TNOR_REG
// End of subcircuit definition
```

16-bit PASS A

```
// Cell name: TNOR_PASSA
// Function: 16-bit PASS
// Inputs: A<15:0> B<15:0>
// Outputs: out<15:0>
subckt TNOR_PASSA VDD VSS in0 in1 in2 in3 in4 in5 in6 in7 in8 in9 in10
in11 \
       in12 in13 in14 in15 control out0 out1 out2 out3 out4 out5 \
       out6 out7 out8 out9 out10 out11 out12 out13 out14 out15
I0 (VDD VSS in0 control out0) TNOR_TX
I1 (VDD VSS in1 control out1) TNOR_TX
I2 (VDD VSS in2 control out2) TNOR_TX
I3 (VDD VSS in3 control out3) TNOR_TX
I4 (VDD VSS in4 control out4) TNOR_TX
I5 (VDD VSS in5 control out5) TNOR_TX
I6 (VDD VSS in6 control out6) TNOR_TX
I7 (VDD VSS in7 control out7) TNOR_TX
I8 (VDD VSS in8 control out8) TNOR_TX
I9 (VDD VSS in9 control out9) TNOR_TX
I10 (VDD VSS in10 control out10) TNOR_TX
I11 (VDD VSS in11 control out11) TNOR_TX
```

```
I12 (VDD VSS in12 control out12) TNOR_TX
I13 (VDD VSS in13 control out13) TNOR_TX
I14 (VDD VSS in14 control out14) TNOR_TX
I15 (VDD VSS in15 control out15) TNOR_TX
ends TNOR_PASSA
// End of subcircuit definition
```

16-bit Adder/Subtractor

```
// Cell Name: TNOR_CStage_MA
// Function: generates Cbar for the Mirror Adder
// Inputs: A B Cin
// Outputs: Cbar
subckt TNOR_CStage_MA VDD VSS A B Cin Cbar
parameters wp=widfac*180n wn=widfac*90n lp=50n ln=50n mult=1 widfac=1

MP0 (net0 A VDD VDD) PMOS_VTL w=2*wp l=lp as=100n*2*wp ad=100n*2*wp
ps=200n+2*wp pd=200n+2*wp m=mult
MP1 (net0 B VDD VDD) PMOS_VTL w=2*wp l=lp as=100n*2*wp ad=100n*2*wp
ps=200n+2*wp pd=200n+2*wp m=mult
MP2 (Cbar Cin net0 VDD) PMOS_VTL w=2*wp l=lp as=100n*2*wp ad=100n*2*wp
ps=200n+2*wp pd=200n+2*wp m=mult
MP3 (net1 A VDD VDD) PMOS_VTL w=2*wp l=lp as=100n*2*wp ad=100n*2*wp
ps=200n+2*wp pd=200n+2*wp m=mult
MP4 (Cbar B net1 VDD) PMOS_VTL w=2*wp l=lp as=100n*2*wp ad=100n*2*wp
ps=200n+2*wp pd=200n+2*wp m=mult

MN5 (net2 A VSS VSS) NMOS_VTL w=2*wn l=ln as=100n*2*wn ad=100n*2*wn
ps=200n+2*wn pd=200n+2*wn m=mult
MN6 (net2 B VSS VSS) NMOS_VTL w=2*wn l=ln as=100n*2*wn ad=100n*2*wn
ps=200n+2*wn pd=200n+2*wn m=mult
MN7 (Cbar Cin net2 VSS) NMOS_VTL w=2*wn l=ln as=100n*2*wn ad=100n*2*wn
ps=200n+2*wn pd=200n+2*wn m=mult
MN8 (net3 A VSS VSS) NMOS_VTL w=2*wn l=ln as=100n*2*wn ad=100n*2*wn
ps=200n+2*wn pd=200n+2*wn m=mult
MN9 (Cbar B net3 VSS) NMOS_VTL w=2*wn l=ln as=100n*2*wn ad=100n*2*wn
ps=200n+2*wn pd=200n+2*wn m=mult
ends TNOR_CStage_MA
// Ends subckt def

// Cell name: TNOR_SStage_MA
// Function: Generates Sbar for Mirror Adder
// Inputs: A B Cin Cbar
// Outputs: Sbar
subckt TNOR_SStage_MA VDD VSS A B Cin Cbar Sbar
parameters wp=widfac*180n wn=widfac*90n lp=50n ln=50n mult=1 widfac=1

MP0 (net0 A VDD VDD) PMOS_VTL w=2*wp l=lp as=100n*2*wp ad=100n*2*wp
ps=200n+2*wp pd=200n+2*wp m=mult
```

```
MP1 (net0 B VDD VDD) PMOS_VTL w=2*wp l=lp as=100n*2*wp ad=100n*2*wp
ps=200n+2*wp pd=200n+2*wp m=mult
MP2 (net0 Cin VDD VDD) PMOS_VTL w=2*wp l=lp as=100n*2*wp ad=100n*2*wp
ps=200n+2*wp pd=200n+2*wp m=mult
MP3 (Sbar Cbar net0 VDD) PMOS_VTL w=2*wp l=lp as=100n*2*wp
ad=100n*2*wp ps=200n+2*wp pd=200n+2*wp m=mult
MP4 (net1 A VDD VDD) PMOS_VTL w=3*wp l=lp as=100n*3*wp ad=100n*3*wp
ps=200n+3*wp pd=200n+3*wp m=mult
MP5 (net2 B net1 VDD) PMOS_VTL w=3*wp l=lp as=100n*3*wp ad=100n*3*wp
ps=200n+3*wp pd=200n+3*wp m=mult
MP6 (Sbar Cin net2 VDD) PMOS_VTL w=3*wp l=lp as=100n*3*wp ad=100n*3*wp
ps=200n+3*wp pd=200n+3*wp m=mult

MN7 (net3 A VSS VSS) NMOS_VTL w=2*wn l=ln as=100n*2*wn ad=100n*2*wn
ps=200n+2*wn pd=200n+2*wn m=mult
MN8 (net3 B VSS VSS) NMOS_VTL w=2*wn l=ln as=100n*2*wn ad=100n*2*wn
ps=200n+2*wn pd=200n+2*wn m=mult
MN9 (net3 Cin VSS VSS) NMOS_VTL w=2*wn l=ln as=100n*2*wn ad=100n*2*wn
ps=200n+2*wn pd=200n+2*wn m=mult
MN10 (Sbar Cbar net3 VSS) NMOS_VTL w=2*wn l=ln as=100n*2*wn
ad=100n*2*wn ps=200n+2*wn pd=200n+2*wn m=mult
MN11 (net4 A VSS VSS) NMOS_VTL w=3*wn l=ln as=100n*3*wn ad=100n*3*wn
ps=200n+3*wn pd=200n+3*wn m=mult
MN12 (net5 B net4 VSS) NMOS_VTL w=3*wn l=ln as=100n*3*wn ad=100n*3*wn
ps=200n+3*wn pd=200n+3*wn m=mult
MN13 (Sbar Cin net5 VSS) NMOS_VTL w=3*wn l=ln as=100n*3*wn
ad=100n*3*wn ps=200n+3*wn pd=200n+3*wn m=mult
ends TNOR_SStage_MA
// Ends subckt def

// Cell name: TNOR_MA
// Function: Mirror Adder
// Inputs: A B Cin
// Outputs: Cbar Sbar
subckt TNOR_MA VDD VSS A B Cin Sel Sbar Cbar
parameters wcf=1 wsf=1
I0 (VDD VSS B Sel net0) TNOR_XOR
I1 (VDD VSS A net0 Cin Cbar) TNOR_CStage_MA widfac=wcf
I2 (VDD VSS A net0 Cin Cbar Sbar) TNOR_SStage_MA widfac=wsf
ends TNOR_MA
// Ends subckt def

// Cell name: TNOR_2bitMA
// Funtion: 2-bit Mirror Adder
// Inputs: A0 A1 B0 B1 Cin
// Outputs: S0 S1 Cout
subckt TNOR_2bitMA VDD VSS A0 A1 B0 B1 Cin Sel S0 S1 Cout
parameters wcf0=1 wsf0=1 wcf1=1 wsf1=1
I0 (VDD VSS A0 B0 Cin Sel S0bar C0bar) TNOR_MA wcf=wcf0 wsf=wsf0
I1 (VDD VSS S0bar S0) TNOR_Inverter
```

```
I2 (VDD VSS A1 A1bar) TNOR_Inverter
I3 (VDD VSS B1 B1bar) TNOR_Inverter
I4 (VDD VSS A1bar B1bar C0bar Sel S1 Cout) TNOR_MA wcf=wcf1 wsf=wsf1
ends TNOR_2bitMA
// Ends subckt def

// Cell name: TNOR_16bitMA
// Function: 16-bit Mirror Adder
// Inputs: A0-A15, B0-B15
// Outputs: S0-S15, Cout
subckt TNOR_16bitMA VDD VSS A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12
A13 A14 A15 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13 B14 B15 Cin
S0 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14 S15 Cout
parameters wfc0=1 wfs0=1 wfc1=1 wfs1=1 wfc2=1 wfs2=1 wfc3=1 wfs3=1
wfc4=1 wfs4=1 wfc5=1 wfs5=1 wfc6=1 wfs6=1 wfc7=1 wfs7=1 wfc8=1 wfs8=1
wfc9=1 wfs9=1 wfc10=1 wfs10=1 wfc11=1 wfs11=1 wfc12=1 wfs12=1 wfc13=1
wfs13=1 wfc14=1 wfs14=1 wfc15=1 wfs15=1

I0 (VDD VSS A0 A1 B0 B1 Cin Cin S0 S1 net0) TNOR_2bitMA wcf0=wfc0
wsf0=wfs0 wcf1=wfc1 wsf1=wfs1
I1 (VDD VSS A2 A3 B2 B3 net0 Cin S2 S3 net1) TNOR_2bitMA wcf0=wfc2
wsf0=wfs2 wcf1=wfc3 wsf1=wfs3
I2 (VDD VSS A4 A5 B4 B5 net1 Cin S4 S5 net2) TNOR_2bitMA wcf0=wfc4
wsf0=wfs4 wcf1=wfc5 wsf1=wfs5
I3 (VDD VSS A6 A7 B6 B7 net2 Cin S6 S7 net3) TNOR_2bitMA wcf0=wfc6
wsf0=wfs6 wcf1=wfc7 wsf1=wfs7
I4 (VDD VSS A8 A9 B8 B9 net3 Cin S8 S9 net4) TNOR_2bitMA wcf0=wfc8
wsf0=wfs8 wcf1=wfc9 wsf1=wfs9
I5 (VDD VSS A10 A11 B10 B11 net4 Cin S10 S11 net5) TNOR_2bitMA
wcf0=wfc10 wsf0=wfs10 wcf1=wfc11 wsf1=wfs11
I6 (VDD VSS A12 A13 B12 B13 net5 Cin S12 S13 net6) TNOR_2bitMA
wcf0=wfc12 wsf0=wfs12 wcf1=wfc13 wsf1=wfs13
I7 (VDD VSS A14 A15 B14 B15 net6 Cin S14 S15 CoutB) TNOR_2bitMA
wcf0=wfc14 wsf0=wfs14 wcf1=wfc15 wsf1=wfs15
I8 (VDD VSS Cin CoutB Cout) TNOR_XOR
ends TNOR_16bitMA
// Ends subckt def
```

16-bit Shifter

```
// Cell Name: TNOR_Shifter
// Function: 16 bit Barrel Shifter, can perform left and right shifts
// Inputs: A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15 B0 B1
D
// Outputs: out0 out1 out2 out3 out4 out5 out6 out7 out8 out9 out10
out11 out12 out13 out14 out15
subckt TNOR_Shifter VDD VSS A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12
A13 A14 A15 B0 B1 D out0 out1 out2 out3 out4 out5 out6 out7 out8 out9
out10 out11 out12 out13 out14 out15
```

Matthew Ridder,
Joseph Fackler,
Doug Etts

```
IO (VDD VSS B0 B1 D hot0 hot1 hot2 hot3 hot4 hot5 hot6 hot7) Decoder

I1 (VDD VSS A1 hot0 buf0) TNOR_TX
I2 (VDD VSS A2 hot0 buf1) TNOR_TX
I3 (VDD VSS A3 hot0 buf2) TNOR_TX
I4 (VDD VSS A4 hot0 buf3) TNOR_TX
I5 (VDD VSS A5 hot0 buf4) TNOR_TX
I6 (VDD VSS A6 hot0 buf5) TNOR_TX
I7 (VDD VSS A7 hot0 buf6) TNOR_TX
I8 (VDD VSS A8 hot0 buf7) TNOR_TX
I9 (VDD VSS A9 hot0 buf8) TNOR_TX
I10 (VDD VSS A10 hot0 buf9) TNOR_TX
I11 (VDD VSS A11 hot0 buf10) TNOR_TX
I12 (VDD VSS A12 hot0 buf11) TNOR_TX
I13 (VDD VSS A13 hot0 buf12) TNOR_TX
I14 (VDD VSS A14 hot0 buf13) TNOR_TX
I15 (VDD VSS A15 hot0 buf14) TNOR_TX
I16 (VDD VSS VSS hot0 buf15) TNOR_TX

I17 (VDD VSS A2 hot1 buf0) TNOR_TX
I18 (VDD VSS A3 hot1 buf1) TNOR_TX
I19 (VDD VSS A4 hot1 buf2) TNOR_TX
I20 (VDD VSS A5 hot1 buf3) TNOR_TX
I21 (VDD VSS A6 hot1 buf4) TNOR_TX
I22 (VDD VSS A7 hot1 buf5) TNOR_TX
I23 (VDD VSS A8 hot1 buf6) TNOR_TX
I24 (VDD VSS A9 hot1 buf7) TNOR_TX
I25 (VDD VSS A10 hot1 buf8) TNOR_TX
I26 (VDD VSS A11 hot1 buf9) TNOR_TX
I27 (VDD VSS A12 hot1 buf10) TNOR_TX
I28 (VDD VSS A13 hot1 buf11) TNOR_TX
I29 (VDD VSS A14 hot1 buf12) TNOR_TX
I30 (VDD VSS A15 hot1 buf13) TNOR_TX
I31 (VDD VSS VSS hot1 buf14) TNOR_TX
I32 (VDD VSS VSS hot1 buf15) TNOR_TX

I33 (VDD VSS A3 hot2 buf0) TNOR_TX
I34 (VDD VSS A4 hot2 buf1) TNOR_TX
I35 (VDD VSS A5 hot2 buf2) TNOR_TX
I36 (VDD VSS A6 hot2 buf3) TNOR_TX
I37 (VDD VSS A7 hot2 buf4) TNOR_TX
I38 (VDD VSS A8 hot2 buf5) TNOR_TX
I39 (VDD VSS A9 hot2 buf6) TNOR_TX
I40 (VDD VSS A10 hot2 buf7) TNOR_TX
I41 (VDD VSS A11 hot2 buf8) TNOR_TX
I42 (VDD VSS A12 hot2 buf9) TNOR_TX
I43 (VDD VSS A13 hot2 buf10) TNOR_TX
I44 (VDD VSS A14 hot2 buf11) TNOR_TX
I45 (VDD VSS A15 hot2 buf12) TNOR_TX
I46 (VDD VSS VSS hot2 buf13) TNOR_TX
```

```
I47 (VDD VSS VSS hot2 buf14) TNOR_TX
I48 (VDD VSS VSS hot2 buf15) TNOR_TX

I49 (VDD VSS A4 hot3 buf0) TNOR_TX
I50 (VDD VSS A5 hot3 buf1) TNOR_TX
I51 (VDD VSS A6 hot3 buf2) TNOR_TX
I52 (VDD VSS A7 hot3 buf3) TNOR_TX
I53 (VDD VSS A8 hot3 buf4) TNOR_TX
I54 (VDD VSS A9 hot3 buf5) TNOR_TX
I55 (VDD VSS A10 hot3 buf6) TNOR_TX
I56 (VDD VSS A11 hot3 buf7) TNOR_TX
I57 (VDD VSS A12 hot3 buf8) TNOR_TX
I58 (VDD VSS A13 hot3 buf9) TNOR_TX
I59 (VDD VSS A14 hot3 buf10) TNOR_TX
I60 (VDD VSS A15 hot3 buf11) TNOR_TX
I61 (VDD VSS VSS hot3 buf12) TNOR_TX
I62 (VDD VSS VSS hot3 buf13) TNOR_TX
I63 (VDD VSS VSS hot3 buf14) TNOR_TX
I64 (VDD VSS VSS hot3 buf15) TNOR_TX

I65 (VDD VSS VSS hot4 buf0) TNOR_TX
I66 (VDD VSS A0 hot4 buf1) TNOR_TX
I67 (VDD VSS A1 hot4 buf2) TNOR_TX
I68 (VDD VSS A2 hot4 buf3) TNOR_TX
I69 (VDD VSS A3 hot4 buf4) TNOR_TX
I70 (VDD VSS A4 hot4 buf5) TNOR_TX
I71 (VDD VSS A5 hot4 buf6) TNOR_TX
I72 (VDD VSS A6 hot4 buf7) TNOR_TX
I73 (VDD VSS A7 hot4 buf8) TNOR_TX
I74 (VDD VSS A8 hot4 buf9) TNOR_TX
I75 (VDD VSS A9 hot4 buf10) TNOR_TX
I76 (VDD VSS A10 hot4 buf11) TNOR_TX
I77 (VDD VSS A11 hot4 buf12) TNOR_TX
I78 (VDD VSS A12 hot4 buf13) TNOR_TX
I79 (VDD VSS A13 hot4 buf14) TNOR_TX
I80 (VDD VSS A14 hot4 buf15) TNOR_TX

I81 (VDD VSS VSS hot5 buf0) TNOR_TX
I82 (VDD VSS VSS hot5 buf1) TNOR_TX
I83 (VDD VSS A0 hot5 buf2) TNOR_TX
I84 (VDD VSS A1 hot5 buf3) TNOR_TX
I85 (VDD VSS A2 hot5 buf4) TNOR_TX
I86 (VDD VSS A3 hot5 buf5) TNOR_TX
I87 (VDD VSS A4 hot5 buf6) TNOR_TX
I88 (VDD VSS A5 hot5 buf7) TNOR_TX
I89 (VDD VSS A6 hot5 buf8) TNOR_TX
I90 (VDD VSS A7 hot5 buf9) TNOR_TX
I91 (VDD VSS A8 hot5 buf10) TNOR_TX
I92 (VDD VSS A9 hot5 buf11) TNOR_TX
I93 (VDD VSS A10 hot5 buf12) TNOR_TX
```

```
I94 (VDD VSS A11 hot5 buf13) TNOR_TX
I95 (VDD VSS A12 hot5 buf14) TNOR_TX
I96 (VDD VSS A13 hot5 buf15) TNOR_TX

I97 (VDD VSS VSS hot6 buf0) TNOR_TX
I98 (VDD VSS VSS hot6 buf1) TNOR_TX
I99 (VDD VSS VSS hot6 buf2) TNOR_TX
I100 (VDD VSS A0 hot6 buf3) TNOR_TX
I101 (VDD VSS A1 hot6 buf4) TNOR_TX
I102 (VDD VSS A2 hot6 buf5) TNOR_TX
I103 (VDD VSS A3 hot6 buf6) TNOR_TX
I104 (VDD VSS A4 hot6 buf7) TNOR_TX
I105 (VDD VSS A5 hot6 buf8) TNOR_TX
I106 (VDD VSS A6 hot6 buf9) TNOR_TX
I107 (VDD VSS A7 hot6 buf10) TNOR_TX
I108 (VDD VSS A8 hot6 buf11) TNOR_TX
I109 (VDD VSS A9 hot6 buf12) TNOR_TX
I110 (VDD VSS A10 hot6 buf13) TNOR_TX
I111 (VDD VSS A11 hot6 buf14) TNOR_TX
I112 (VDD VSS A12 hot6 buf15) TNOR_TX

I113 (VDD VSS VSS hot7 buf0) TNOR_TX
I114 (VDD VSS VSS hot7 buf1) TNOR_TX
I115 (VDD VSS VSS hot7 buf2) TNOR_TX
I116 (VDD VSS VSS hot7 buf3) TNOR_TX
I117 (VDD VSS A0 hot7 buf4) TNOR_TX
I118 (VDD VSS A1 hot7 buf5) TNOR_TX
I119 (VDD VSS A2 hot7 buf6) TNOR_TX
I120 (VDD VSS A3 hot7 buf7) TNOR_TX
I121 (VDD VSS A4 hot7 buf8) TNOR_TX
I122 (VDD VSS A5 hot7 buf9) TNOR_TX
I123 (VDD VSS A6 hot7 buf10) TNOR_TX
I124 (VDD VSS A7 hot7 buf11) TNOR_TX
I125 (VDD VSS A8 hot7 buf12) TNOR_TX
I126 (VDD VSS A9 hot7 buf13) TNOR_TX
I127 (VDD VSS A10 hot7 buf14) TNOR_TX
I128 (VDD VSS A11 hot7 buf15) TNOR_TX

I129 (VDD VSS buf0 out0) InvBuffer
I130 (VDD VSS buf1 out1) InvBuffer
I131 (VDD VSS buf2 out2) InvBuffer
I132 (VDD VSS buf3 out3) InvBuffer
I133 (VDD VSS buf4 out4) InvBuffer
I134 (VDD VSS buf5 out5) InvBuffer
I135 (VDD VSS buf6 out6) InvBuffer
I136 (VDD VSS buf7 out7) InvBuffer
I137 (VDD VSS buf8 out8) InvBuffer
I138 (VDD VSS buf9 out9) InvBuffer
I139 (VDD VSS buf10 out10) InvBuffer
I140 (VDD VSS buf11 out11) InvBuffer
```

```
I141 (VDD VSS buf12 out12) InvBuffer
I142 (VDD VSS buf13 out13) InvBuffer
I143 (VDD VSS buf14 out14) InvBuffer
I144 (VDD VSS buf15 out15) InvBuffer
ends TNOR_Shifter
// Ends subckt def
```

16-bit AND

```
// Cell name: TNOR_AND
// Function: 16-bit AND
// Inputs: A<15:0> B<15:0>
// Outputs: out<15:0>
subckt TNOR_AND VDD VSS A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 \
       A12 A13 A14 A15 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 \
       B11 B12 B13 B14 B15 out0 out1 out2 out3 out4 out5 \
       out6 out7 out8 out9 out10 out11 out12 out13 out14 out15
I0 (VDD VSS A0 B0 out0) TNOR_AND2
I1 (VDD VSS A1 B1 out1) TNOR_AND2
I2 (VDD VSS A2 B2 out2) TNOR_AND2
I3 (VDD VSS A3 B3 out3) TNOR_AND2
I4 (VDD VSS A4 B4 out4) TNOR_AND2
I5 (VDD VSS A5 B5 out5) TNOR_AND2
I6 (VDD VSS A6 B6 out6) TNOR_AND2
I7 (VDD VSS A7 B7 out7) TNOR_AND2
I8 (VDD VSS A8 B8 out8) TNOR_AND2
I9 (VDD VSS A9 B9 out9) TNOR_AND2
I10 (VDD VSS A10 B10 out10) TNOR_AND2
I11 (VDD VSS A11 B11 out11) TNOR_AND2
I12 (VDD VSS A12 B12 out12) TNOR_AND2
I13 (VDD VSS A13 B13 out13) TNOR_AND2
I14 (VDD VSS A14 B14 out14) TNOR_AND2
I15 (VDD VSS A15 B15 out15) TNOR_AND2
ends TNOR_AND
// End of subcircuit definition
```

16-bit OR

```
// Cell name: TNOR_OR
// Function: 16-bit OR
// Inputs: A<15:0> B<15:0>
// Outputs: out<15:0>
subckt TNOR_OR VDD VSS A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 \
       A12 A13 A14 A15 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 \
       B11 B12 B13 B14 B15 out0 out1 out2 out3 out4 out5 \
       out6 out7 out8 out9 out10 out11 out12 out13 out14 out15
I0 (VDD VSS A0 B0 out0) TNOR_OR2
I1 (VDD VSS A1 B1 out1) TNOR_OR2
I2 (VDD VSS A2 B2 out2) TNOR_OR2
I3 (VDD VSS A3 B3 out3) TNOR_OR2
```

```
I4 (VDD VSS A4 B4 out4) TNOR_OR2
I5 (VDD VSS A5 B5 out5) TNOR_OR2
I6 (VDD VSS A6 B6 out6) TNOR_OR2
I7 (VDD VSS A7 B7 out7) TNOR_OR2
I8 (VDD VSS A8 B8 out8) TNOR_OR2
I9 (VDD VSS A9 B9 out9) TNOR_OR2
I10 (VDD VSS A10 B10 out10) TNOR_OR2
I11 (VDD VSS A11 B11 out11) TNOR_OR2
I12 (VDD VSS A12 B12 out12) TNOR_OR2
I13 (VDD VSS A13 B13 out13) TNOR_OR2
I14 (VDD VSS A14 B14 out14) TNOR_OR2
I15 (VDD VSS A15 B15 out15) TNOR_OR2
ends TNOR_OR
// End of subcircuit definition
```

16-bit 8-1 MUX

```
// Cell name: TNOR_MUX
// Function: 16-bit 8-1 Mux
// Inputs: A<15:0> B<15:0> C<15:0> D<15:0> E<15:0> F<15:0> G<15:0>
H<15:0> S0 S1       S2
// Outputs: out<15:0>
subckt TNOR_MUX VDD VSS A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13
A14 A15 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13 B14 B15 C0 C1 C2
C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 D0 D1 D2 D3 D4 D5 D6 D7
D8 D9 D10 D11 D12 D13 D14 D15 E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 E10 E11
E12 E13 E14 E15 F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15
G0 G1 G2 G3 G4 G5 G6 G7 G8 G9 G10 G11 G12 G13 G14 G15 H0 H1 H2 H3 H4
H5 H6 H7 H8 H9 H10 H11 H12 H13 H14 H15 S0 S1 S2 out0 out1 out2 out3
out4 out5 out6 out7 out8 out9 out10 out11 out12 out13 out14 out15

I0 (VDD VSS A0 B0 C0 D0 E0 F0 G0 H0 S0 S1 S2 out0) TNOR_MUX8to1
I1 (VDD VSS A1 B1 C1 D1 E1 F1 G1 H1 S0 S1 S2 out1) TNOR_MUX8to1
I2 (VDD VSS A2 B2 C2 D2 E2 F2 G2 H2 S0 S1 S2 out2) TNOR_MUX8to1
I3 (VDD VSS A3 B3 C3 D3 E3 F3 G3 H3 S0 S1 S2 out3) TNOR_MUX8to1
I4 (VDD VSS A4 B4 C4 D4 E4 F4 G4 H4 S0 S1 S2 out4) TNOR_MUX8to1
I5 (VDD VSS A5 B5 C5 D5 E5 F5 G5 H5 S0 S1 S2 out5) TNOR_MUX8to1
I6 (VDD VSS A6 B6 C6 D6 E6 F6 G6 H6 S0 S1 S2 out6) TNOR_MUX8to1
I7 (VDD VSS A7 B7 C7 D7 E7 F7 G7 H7 S0 S1 S2 out7) TNOR_MUX8to1
I8 (VDD VSS A8 B8 C8 D8 E8 F8 G8 H8 S0 S1 S2 out8) TNOR_MUX8to1
I9 (VDD VSS A9 B9 C9 D9 E9 F9 G9 H9 S0 S1 S2 out9) TNOR_MUX8to1
I10 (VDD VSS A10 B10 C10 D10 E10 F10 G10 H10 S0 S1 S2 out10)
TNOR_MUX8to1
I11 (VDD VSS A11 B11 C11 D11 E11 F11 G11 H11 S0 S1 S2 out11)
TNOR_MUX8to1
I12 (VDD VSS A12 B12 C12 D12 E12 F12 G12 H12 S0 S1 S2 out12)
TNOR_MUX8to1
I13 (VDD VSS A13 B13 C13 D13 E13 F13 G13 H13 S0 S1 S2 out13)
TNOR_MUX8to1
```

```
I14 (VDD VSS A14 B14 C14 D14 E14 F14 G14 H14 S0 S1 S2 out14)
TNOR_MUX8to1
I15 (VDD VSS A15 B15 C15 D15 E15 F15 G15 H15 S0 S1 S2 out15)
TNOR_MUX8to1
ends TNOR_MUX
//End of subcircuit definition
```

ALU Top-Level Connections

```
// Cell Name: TNOR_ALU
// Function: 16 bit ALU, top level connections with registers
// Inputs: A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15 B0 B1
B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13 B14 B15 con0 con1 con2 CLK
// Outputs: out0 out1 out2 out3 out4 out5 out6 out7 out8 out9 out10
out11 out12 out13 out14 out15 cout
subckt TNOR_ALU VDD VSS A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13
A14 A15 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13 B14 B15 LRin
con0 con1 con2 CLK out0 out1 out2 out3 out4 out5 out6 out7 out8 out9
out10 out11 out12 out13 out14 out15 cout

I0 (VDD VSS A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15 CLK
Ain0 Ain1 Ain2 Ain3 Ain4 Ain5 Ain6 Ain7 Ain8 Ain9 Ain10 Ain11 Ain12
Ain13 Ain14 Ain15) TNOR_REG

I1 (VDD VSS B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13 B14 B15 CLK
Bin0 Bin1 Bin2 Bin3 Bin4 Bin5 Bin6 Bin7 Bin8 Bin9 Bin10 Bin11 Bin12
Bin13 Bin14 Bin15) TNOR_REG

I2 (VDD VSS aluout0 aluout1 aluout2 aluout3 aluout4 aluout5 aluout6
aluout7 aluout8 aluout9 aluout10 aluout11 aluout12 aluout13 aluout14
aluout15 CLK muxA0 muxA1 muxA2 muxA3 muxA4 muxA5 muxA6 muxA7 muxA8
muxA9 muxA10 muxA11 muxA12 muxA13 muxA14 muxA15) TNOR_REG

I3 (VDD VSS A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15 con0
muxB0 muxB1 muxB2 muxB3 muxB4 muxB5 muxB6 muxB7 muxB8 muxB9 muxB10
muxB11 muxB12 muxB13 muxB14 muxB15) TNOR_PASSA

I4 (VDD VSS A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15 B0
B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13 B14 B15 con0 muxC0 muxC1
muxC2 muxC3 muxC4 muxC5 muxC6 muxC7 muxC8 muxC9 muxC10 muxC11 muxC12
muxC13 muxC14 muxC15 cout) TNOR_16bitMA

I5 (VDD VSS A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15 B0
B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13 B14 B15 con0 muxD0 muxD1
muxD2 muxD3 muxD4 muxD5 muxD6 muxD7 muxD8 muxD9 muxD10 muxD11 muxD12
muxD13 muxD14 muxD15 cout) TNOR_16bitMA
```

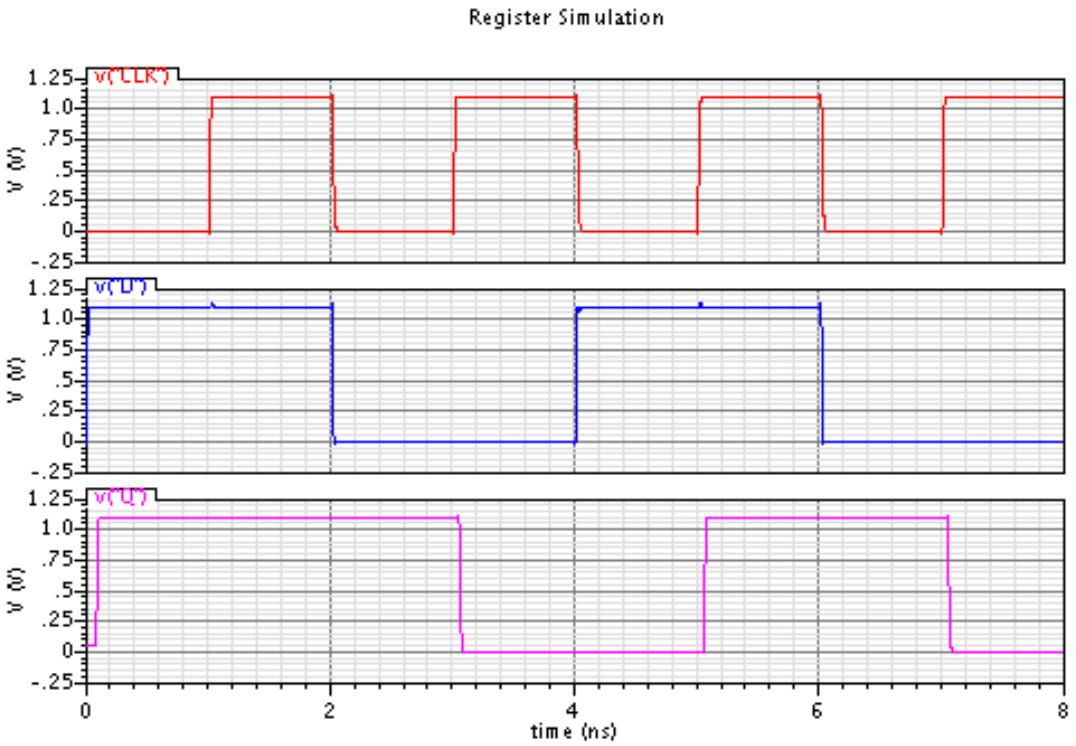
```
I6 (VDD VSS A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15 B0
B1 LRin muxE0 muxE1 muxE2 muxE3 muxE4 muxE5 muxE6 muxE7 muxE8 muxE9
muxE10 muxE11 muxE12 muxE13 muxE14 muxE15) TNOR_Shifter

I7 (VDD VSS A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15 B0
B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13 B14 B15 muxF0 muxF1 muxF2
muxF3 muxF4 muxF5 muxF6 muxF7 muxF8 muxF9 muxF10 muxF11 muxF12 muxF13
muxF14 muxF15) TNOR_AND

I8 (VDD VSS A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15 B0
B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13 B14 B15 muxG0 muxG1 muxG2
muxG3 muxG4 muxG5 muxG6 muxG7 muxG8 muxG9 muxG10 muxG11 muxG12 muxG13
muxG14 muxG15) TNOR_OR

I9 (VDD VSS muxA0 muxA1 muxA2 muxA3 muxA4 muxA5 muxA6 muxA7 muxA8
muxA9 muxA10 muxA11 muxA12 muxA13 muxA14 muxA15 muxB0 muxB1 muxB2
muxB3 muxB4 muxB5 muxB6 muxB7 muxB8 muxB9 muxB10 muxB11 muxB12 muxB13
muxB14 muxB15 muxC0 muxC1 muxC2 muxC3 muxC4 muxC5 muxC6 muxC7 muxC8
muxC9 muxC10 muxC11 muxC12 muxC13 muxC14 muxC15 muxD0 muxD1 muxD2
muxD3 muxD4 muxD5 muxD6 muxD7 muxD8 muxD9 muxD10 muxD11 muxD12 muxD13
muxD14 muxD15 muxE0 muxE1 muxE2 muxE3 muxE4 muxE5 muxE6 muxE7 muxE8
muxE9 muxE10 muxE11 muxE12 muxE13 muxE14 muxE15 muxF0 muxF1 muxF2
muxF3 muxF4 muxF5 muxF6 muxF7 muxF8 muxF9 muxF10 muxF11 muxF12 muxF13
muxF14 muxF15 muxG0 muxG1 muxG2 muxG3 muxG4 muxG5 muxG6 muxG7 muxG8
muxG9 muxG10 muxG11 muxG12 muxG13 muxG14 muxG15 muxH0 muxH1 muxH2
muxH3 muxH4 muxH5 muxH6 muxH7 muxH8 muxH9 muxH10 muxH11 muxH12 muxH13
muxH14 muxH15 con0 con1 con2 aluout0 aluout1 aluout2 aluout3 aluout4
aluout5 aluout6 aluout7 aluout8 aluout9 aluout10 aluout11 aluout12
aluout13 aluout14 aluout15) TNOR_MUX

I10 (VDD VSS aluout0 aluout1 aluout2 aluout3 aluout4 aluout5 aluout6
aluout7 aluout8 aluout9 aluout10 aluout11 aluout12 aluout13 aluout14
aluout15 CLK out0 out1 out2 out3 out4 out5 out6 out7 out8 out9 out10
out11 out12 out13 out14 out15) TNOR_REG

ends TNOR_ALU
//Ends subcircuit definition
```

Matthew Ridder,
Joseph Fackler,
Doug Etts

Simulation Results

Register Simulation Results

Here is the simulation for a 1-bit register. Since the 16-bit register is just 16 1-bit registers in parallel, this simulation can be extended for all 16 bits:



Add/Sub Simulation Results

Here is a table we generated for a 1-bit add/sub block. Cin=0 represents an addition, and Cin=1 represents a subtraction:

Cin	A	B	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	0	1
1	1	1	0	0

For additions, Cout represents the carry bit. For subtractions, Cout acts as a sign flag to indicate positive or negative numbers. (0-> positive, 1-> negative)

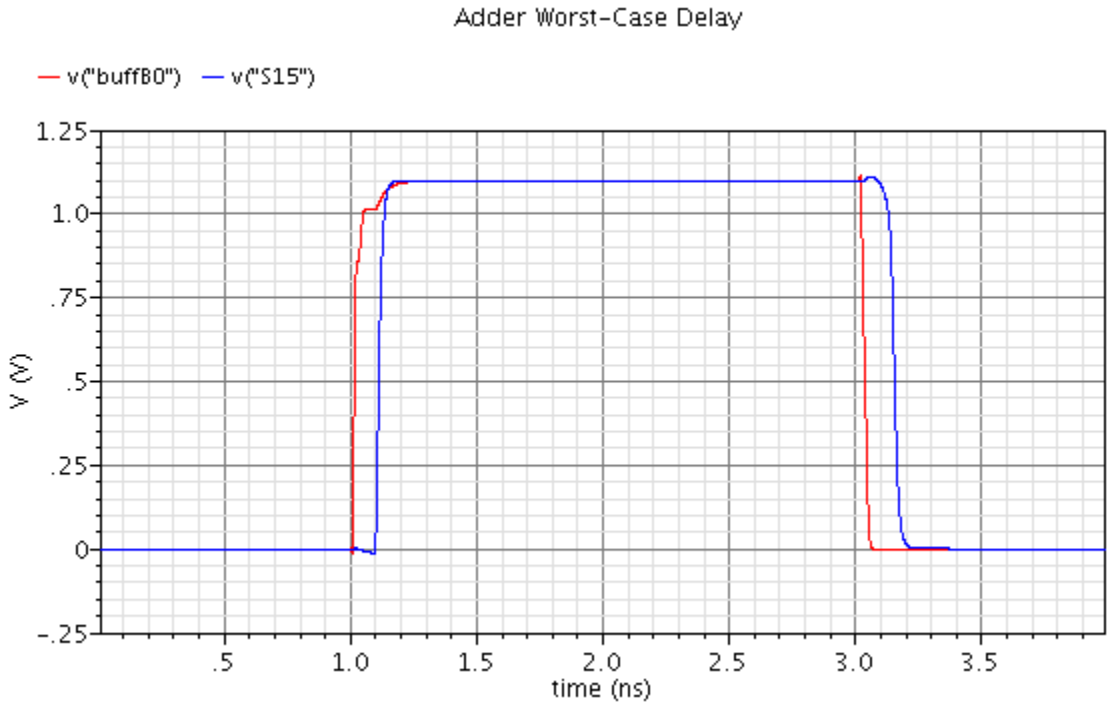
Matthew Ridder,
Joseph Fackler,
Doug Etts

Here is a table we generated for our 16-bit add/sub block:

A	B	Cin	Cout	S
0x7fff	0x7fff	0	0	0xfff7
0x7fff	0x0000	0	0	0x7fff
0x9999	0x9911	0	1	0x32aa
0x9999	0x9911	1	0	0x0088
0x0000	0x8000	1	1	0x8000
0x7fff	0x7fff	1	0	0x0000

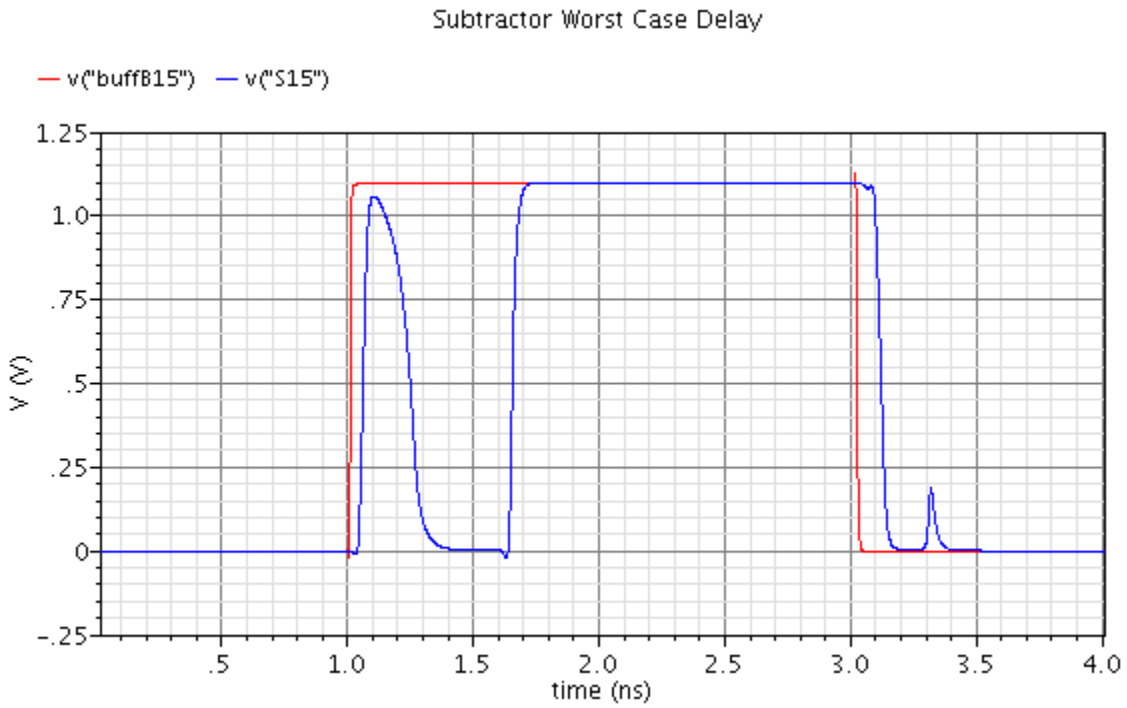
As you can see from the table, we tested a few different cases to ensure that our add/sub block worked properly. For the add part of the block, we tested if it could add 2 positive numbers without generating a carry, 2 positive numbers where one of the numbers was 0, and 2 positive numbers that generated a carry. For the sub part of the block, we tested if it could subtract a smaller number from a bigger number to get a positive result, a bigger number from a smaller number to get a negative result, and the same number to get 0.

After testing the functionality of the add/sub block, we found the worst case delay for both the add and sub parts of it. In this simulation, we set A = 0x7fff and B = 0x0001 as our worst case scenario for the add part of the block. We then measured the delay between when bit B<0> rises to 1 and the last sum bit (S15) rises to 1 after the carry bit propagates through the add/sub stages.



Matthew Ridder,
Joseph Fackler,
Doug Etts

In this next simulation, we set A = 0x0000 and B = 0x8000 as our worst case scenario for the sub part of our add/sub block. We then measured the delay between when bit B<0> rises to 1 and the last sum bit (S15) rises to 1 after the carry bit propagates through the add/sub stages. The weird signal behavior in S15 between 1 and 1.5ns is just a result of the XOR gate that we are using in the add/sub block. As you can see the sum bit doesn't actually settle until about 1.8ns.



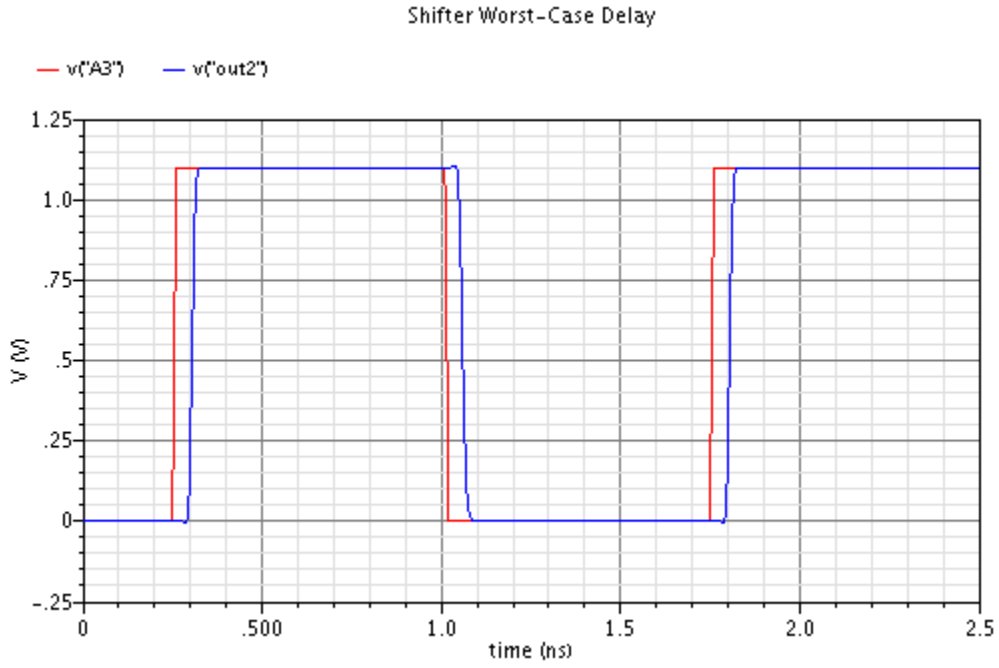
Shifter Simulation Results

Here is a table we generated for our 16-bit shifter. Bits B1 and B0 determine the shift count and bit D determines the direction of the shift. D=0 represents a right shift and D=1 represents a left shift:

Function	A	D	B1	B0	Out
Right Shift 1	1001 1001 1001 1001	0	0	0	0100 1100 1100 1100
Right Shift 2	1001 1001 1001 1001	0	0	1	0010 0110 0110 0110
Right Shift 3	1001 1001 1001 1001	0	1	0	0001 0011 0011 0011
Right Shift 4	1001 1001 1001 1001	0	1	1	0000 1001 1001 1001
Left Shift 1	1001 1001 1001 1001	1	0	0	0011 0011 0011 0010
Left Shift 2	1001 1001 1001 1001	1	0	1	0110 0110 0110 0100
Left Shift 3	1001 1001 1001 1001	1	1	0	1100 1100 1100 1000
Left Shift 4	1001 1001 1001 1001	1	1	1	1001 1001 1001 0000

Matthew Ridder,
Joseph Fackler,
Doug Etts

After we tested the functionality of our shifter, we measured its worst case delay. Because we used a Barrel Shifter implementation, each bit only has to go through a max of 1 transmission gate and a buffer. This means that the delay is about the same for all cases. In this simulation we measured the delay when a bit in position A3 is shifted 1 bit to the right and gets placed in position out2:



ALU Top Level Connectivity Simulation Results

For our ALU connectivity simulation, we iterated through multiple control points with the same A and B values to test that all of the components functioned correctly when connected all together. Here is the table that includes our simulation results for the ALU:

Function	A	B	Control	Out	Cout
PASS A	1001 1001 1001 1001	1001 1001 0001 0001	001	1001 1001 1001 1001	0
ADD	1001 1001 1001 1001	1001 1001 0001 0001	010	0011 0010 1010 1010	1
SUB	1001 1001 1001 1001	1001 1001 0001 0001	011	0000 0000 1000 1000	0
SHIFT	1001 1001 1001 1001	1001 1001 0001 0001	100	0010 0110 0110 0110	1
AND	1001 1001 1001 1001	1001 1001 0001 0001	101	1001 1001 0001 0001	0
OR	1001 1001 1001 1001	1001 1001 0001 0001	110	1001 1001 1001 1001	1

Matthew Ridder,
Joseph Fackler,
Doug Etts

Here are the same simulation results for the ALU but in Hex notation:

Function	A	B	Control	Out	Cout
PASS A	0x9999	0x9911	001	0x9999	0
ADD	0x9999	0x9911	010	0x32aa	1
SUB	0x9999	0x9911	011	0x0088	0
SHIFT	0x9999	0x9911	100	0x2666	1
AND	0x9999	0x9911	101	0x9911	0
OR	0x9999	0x9911	110	0x9999	1

Summary of the Worst-Case Delays for our Components

Operation	**Worst-Case Delay**	**Scenario**
ADD	1.07E-10 s	A=0x0001, B=0x7FFF. The carry bit has to propagate through each stage.
SUB	5.66811E-10 s	A=0x0000, B=0x8000. The carry bit has to propagate through each stage.
SHIFT	4.685382e-11 s	Any size shift in any direction because each bit only has to go through a max of 1 transmission gate and a buffer.
AND	2.991659e-11 s	A=1, B=1. The NMOS transistors are turned on and they have smaller widths than the PMOS transistors.
OR	2.618116e-11 s	A=1, B=0. A single NMOS transistor is turned on in the parallel arrangement in the pull-down network.
PASS A	3.491805e-12 s	Pass either 0 or 1. The delay should be about the same.